**Java Questions**

**Encapsulation:**

it is also known as data hiding .

the main purpose of Encapsulation is used to hide the data from the external class and package and protect the data from accessing the data directly.

we can't access the fields and manipulate it directly. Inorder to do those we use the getter and setter methods to do those things.

**Abstraction:**

It is the process of hiding the internal details and display only its functionality.

Eg:

we know how to shift the gear but we don't know how internally the gear shifting operation works.

**Inheritance:**

It is the process of acquiring the properties of previously declared methods and we can inherit the properties from those classes.

**Diamond Problem in java:**

When a particular class wants to extend one or more class / or a class has the properties that belongs to two or more class then its not possible. we can't extend two or more more classes

**Solution:**

To overcome this we can use inheritance and use impplements in the class.

**Synchronized Keyword:**

When two or more methods try to access the same varaiable, object or resource it leads to interference and memory errors or concurrency.

inorder to overcom this problems we can use the synchronized keywords.the other will pause its work untill the current thread execution is getting completed.

**Transit Keyword :**

the transient keyword is used in java serialazation concept.

Serialization : the process of converting object to byte stream(platform independent). Deserialization: the process of converting byte stream to object.

When we decalre a particular field or variable as transient the particular will not be serialized(the particular data will not converted into byte stream).

**Concrete Class:**

It is a tye of extended or imlemented class where the class has all its implementation of the extended and implemented class.

**Immutable class:**

A class i which its fields can not be changed or optimised is called immmutable class. the immutable class have private and final keywords and modifiers.

**Memory Management in Java:**

The process of allocating and freeing up the space for the objects.

Java does memory management automatically with the help of garbage collector.

When JVM is running the objects are created in the heap which is dedicated for the program.

Whenever garbage collector finds the unused objects it will delete them automatically.

{

Heap-It is an area of memory that is shared among all the threads running in the JVM.

Stack-It stores temporary variables and reference. whenever it function gets completed it will destroy that keyword (or)reference.

}

**MicroService:**

It is an application style that develops an application as a set of small services. Each service runs it own process .

**Wrapper class:**

Wrapper class is used when we need to use primitive as an object in methods.

It has methods to unwrap the object and it will give it back as primitive types.

The most important use of wrappers class is the collection framework works only with objects and it does not support primitive types.

We can create wrapper class object by using

->constructors

->instance of methods

**AutoBoxing:**

It is process of automatic conversion a primitive into a wrapper class object.

**Unboxing:**

It is process of converting primitive class to its respective primitive type.

**Fail Fast:**

Fail fast condition is developing an application in such a way that whenever a failure or fault occurs it will immediately stops and through exception.

Eg:

If one more thread doing modification on collection it through concurrent modification exception.

**Fail Safe:**

Fail safe condition is developing an application in such a way that it can bypass the failure or fault if it is not critical error.

Eg:

CopyonArrayList and ConcurrentHashMap follows fail safe mechanism as it works only on the copy of the object created

**Helper Class:**

It is a type of java class where it contains some basic error handling and helper functions. The functions present in that method are mostly static. The functions in that method contains functions which are not present in the library functions.

**Wrapper Class**:

It will provide the way to represent the value of primitive data type in the form of object.

**Shallow Copy vs Deep Copy:**

**Shallow copy:**

When two reference points to the same class. The change in one reference will also affects the other such type of copy is called as Shallow copy.

**Deep Copy:**

When two reference points to the same class. The change in one reference will not affect the other such type of copy is called as Deep copy.

**Runnable and Callable interface:**

**Runnable:**

It is a type of functional interface and it has only one method run() and it doesn’t return any value.

It is mostly used to work with asynchronous type of programming.

**Callable:**

It is also a type of functional interface but it can return value. It has method call() which can used to handle the exceptions.

**EXceptionHandling in java:**

🡪First create a custom exception

🡪Then create a body for the exception

🡪Then implement the exception in the service layer

**Logging :**

It helps in monitoring the flow of the application. When issues or error occur, it will be useful for finding the problem.

It helps to monitor the application behavior in development, testing and production.

It helps to maintain a healthy and reliable application .which helps to maintain and manage the software.

**Info, warn and error** are a specific type of log which are available with spring starter dependency.

Debug and trace are the logging which are available by adding external dependency.

(Sl4J and logging)

To set different color for different logs we can use the below properties in the application properties.

**Spring.output.ansi.enabled=always**

we can also save the logs in a separate file so that we can it for analysis purpose.

To make it to save it in a file we can use the below property.

**Logger.file.name=LoggerFile.log**

**TESTING:**

**Junit is used to test api while doing testing with api we need to create object for certain places for such type of condition mockito.**

Unit testing is used to check whether the developed code is working properly without any flaws.

We have to write the test case in the test folder of the project.

Every method in the test class must be annotated with @Test to mention that it is referring to a test case.

Every test case follows this format **AAA**

**Arrange**

**Act**

**Assertion**

Arrange :

Initialize the variable of the test case or assign values to the variables.

Act:

Do operations with the variables

Assertions:

Check the final action output with the desired output.

@Extendedwith(MockitoExtension.class)

Whenever we use @Mock and @InjectMock annotataion in our application we have to annotate that class with this annotation.

@Mock:

It creates a implementation of the things or class that are needed for the test to happen.

@InjectMock

It injects the Mocks into the annotated class that requires the mock to perform test operation.

@Test

It helps to identify this method is a test method

@Before

It starts the method before starting each test method

@After

It starts the method after completion of esach test method

@Beforeclass

It starts the method

**Service discovery:**

In microservice kind of architecture. When one service want to connect with another it will not be easy in microservice type of architecture. To make contact with another we need to find whether the service is working or not and finds its service port, to find this we need to use this

Service discovery to use find the service and make connection to the service.

It is classified into two types

Client side service discovery

Server side service discovery

**Server side service discovery:**

when client wants to connect with one of the service, the connection will happen with the help of load balancer and service discovery.

Service discovery holds the details about all the available services in the server and stores it in the form of map in the service discovery.

The service discovery provides the particular detail with the port number to the load balancer and the load balancer will do the needful .

This is called server side service discovery.

**NGNIX, AWS ELB**

**Client side service discovery:**

It is similar to server side service discovery.

But in this the client itself act as a load balancer so that it can itself fetch what service I have to connect.

This is called client side service discovery.

**Netflix Eureka, Zookeeper, Consul**

**WEB STORAGE:**

It stores data in the form of key value pair in the browser. It stores data in the client side browser. The data stored in the browser will stay there and it cannot be transferred anywhere.

Two types of storage:

Local storage

Session storage

LOCAL STORAGE: in local storage the data will be stored in the browser. It cannot be deleted until it is deleted manually. Closing the browser won’t delete the data in the browser.

SESSION STORAGE:

In session storage the data is stored in the browser. The session storage size depends on the browser and system memory data stored in the browser will remain there until the browser is closed.

**Redis**

**Remote Dictionary Server**

It is a type of data structure type of storage where it is used for caching. It is a type of nosql database.

Spring boot is not having any inbuilt cache memory so it depends on third parties like redis,caffine and some others to implement caching in the application.

@EnableCaching

To enable the cahing mechanism in our application we use this annotation above the application class

@CacheConfig

It stores the data from database on the first hit after that it will give the data from the database.

@CachePut

Whenever we updated the data in the database the same data will get updated in the cache storage

@CacheEvict

Whenever we delete any data from the database the particular data will be deleted from the database same as in the cache storage

@Cacheable

Whenever we are using more than one cache process in the method we can use this annotation above that class it has properties like cachename,cachekey etc.,

**Feign Client:**

**API:**

Application Program Interface is a way to communicate between different services over internet.

It is a set of tool that allows the developer to send and receive information between server and client usually in the form of JSON.

**RESTTEMPLATE:**

It is a springclass for client side Http access and maping the response for us. It is synchronous type of REST client

**WEBCLIENT**

It is asynchronous type of Rest invoker. When we compare it with feign client and restemplate those are synchronous type. When resttemplate or feign is invoked in the application the particular method will block other methods until the method working gets completed.

Whereas in webclient it performs asynchronous type it will not block other threads until its execution gets complete.

**Feign Client:**

It is also known as open feign client or declarative rest client.

Feign client is mostly used to consume REST API endpoints which are exposed by third party or micro service.

**Design Patterns:**

It helps to overcome the architectural and design problems which occurred before .

It speeds up the development process and it provides tested and prove-n development models.

**Patterns:**

The patterns won’t generate ready code for us .

It provides a template to follow.

It is a series of formalized best practices tested and verified by the developers before us.

There are 23 different design patterns. And these 23 design patterns are classified into three types

Creational🡪 it deals with the creation of objects.

Structural 🡪it deals with how the classes are actually designed and it also deals with how

Inheritance and composition can provide extra flexibility

Behavioural🡪 it deals with the communication and assignment responsibilities between the classes.

**Singleton Design Pattern:**

It is a type of creational design pattern.

It ensures only one instance of its kind exist.

It provides  single point of access to the application from other part.

It uses the same idea like global variable it let you access the object from anywhere in the application and it also encapsulate the attributes of the object and it ensures that there is only one instance of the class is available.

Singleton pattern is used for logging, drivers objects, caching, and thread pool.

To implement a singleton pattern, we have different approaches, but all of them have the following common concepts.

Private constructor to restrict instantiation of the class from other classes.

Private static variable of the same class that is the only instance of the class.

Public static method that returns the instance of the class, this is the global access point for the outer world to get the instance of the singleton class.

**Eager initialisation**

**static block initialization**

**lazy initialization**

**Thread safe singleton**

**bill pugh singleton--> it is the most widely used method.**

**Advantage:**main advantage of singleton is it saves memory

**Factory Method Design Pattern:**

It is a type of creational design pattern .

**As from the name itself we can understood like that this helps in creating appropriate objects. It is the best way to create an object where object creation logic is hidden from the client.**

**The abstract class or parent class decides which child or sub class it has to initiate.**

It loosens the coupling of given code by separating the products construction code from the code that uses this product.

The factory method depends on inheritance

**Uses:**

If we don’t have any idea of each types of object your code will work with

It makes easy to extend the product construction code independently from the rest of the application.

Allows to introduce new products without breaking the existing code.

Centralizes the product creation code in one place in the program.

**ABSTRACT FACTORY PATTERN:**

Abstract Factory Pattern says that just **define an interface or abstract class for creating families of related (or dependent) objects but without specifying their concrete sub-classes.**That means Abstract Factory lets a class returns a factory of classes. So, this is the reason that Abstract Factory Pattern is one level higher than the Factory Pattern.

An Abstract Factory Pattern is also known as **Kit.**

**Design Principles:**

the design principles are the set of advice used as rules in design making. In Java, the design principles are similar to the design patterns concept. The only difference between the design principle and design pattern is that the design principles are more generalized and abstract. The design pattern contains much more practical advice and concrete.

The most important design principles are SOLID design principles.

S-single responsipility principle

O-Open/close principle

L-Liskov substitution principle

I-Interface segregation principle

D-dependency inversion principle

**SINGLE RESPONSIBILITY PRINCIPLE:**

One class should have one and only reasonability.

Consider a model or entity class that it should represent only one of the class in the application.

By using this type of principle there will be more flexibility to make change in the class, without making changes into the another class.

**OPEN/CLOSED PRINCIPLE**

Software components should be open for extension, but closed for modification.

It is similar to abstract class when we want to use that class we can use that by extending the abstract class in some other class. If we want to make change in any of the abstract method we can’t do it directly on the abstract class we can do the modification by doing it in the extended class.

**LISKOV’S SUBSTITUTION PRINCIPLE:**

Derived types must be completely substitutable for their base types.

It is similar to inheritance. When we develop some class and someone extends our created class and they have to design in such a way that it has to fit in the application without fail.

**INTERFACE SEGREGATION PRINCIPLE:**

Clients should not be forced to implement unnecessary methods which they will not use

Consider I am writing an interface class which has two methods. But the client requires only one method of the interface. Whenever he wants to use this interface anywhere he has to implement both the methods which has no use for one of the method.

This type of creations are not approached so we have to design only based on the necessities or we can do that by creating separate interface for each method so that they can use that whenever they need that.

**DEPENDENCY INVERSION PRINCIPLE:**

Depends on abstraction not on concretion.

We should design our software in such a way that each module should be separate from each other and using the abstract layer we can bind them together